CHARM
Architectural Design

Child Health Advanced Records Management (CHARM)
Utah Department of Health

Preparation of this publication was supported by a contract from
All Kids Count, a program of The Robert Wood Johnson Foundation.

CHARM
Architectural Design

Stephen W. Clyde, Ph.D.

Child Health Advanced Records Management (CHARM)
Utah Department of Health

September 2002

This publication was supported by a contract from All Kids Count, a program of The Robert Wood Johnson Foundation, to the Utah Department of Health. The views, content and citations reflect those of the Utah Department of Health.

Ordering Information
This publication is available online at the All Kids Count web site, www.allkidscount.org.

Table of Contents

1. Introduction

Over the past six years, the Utah Department of Health (UDOH) has developed and deployed a number of health-care information systems, including: Heel-Stick (metabolic) Screening, Utah Statewide Immunization Information System (USIIS), Early Hearing Detection and Intervention (EHDI), Vital Statistics (VS), Birth Defects (BD), and Women, Infant, and Children (WIC). Utah also is in the process of developing an Early Intervention (EI) information system. Each of these systems belongs to a corresponding health-care program, which is responsible for managing that data.

With the exception of some rudimentary links between a few of these information systems, they currently operate independently. Tighter integration among them, however, would improve UDOH's service quality by allowing users of one system to have immediate access to information currently found only in another. For example, EHDI users would improve their follow-up efficiency if they had immediate access to contact information for a child's primary-care provider available in USIIS. Similarly, EI users would benefit from hearing screening data in EHDI, and WIC users could benefit from immunization information in USIIS. Without exception, users of every health-care information system could benefit in some way from access to data in at least one other.

Child Health Advanced Record Management (CHARM) is a concerted effort to share data among health-care information systems in real-time. The shared data for a child constitutes a virtual health-care record that we call the *Child-Health Profile* (CHP). No single health-care system stores or manages complete CHP's. Instead, each system continues to manage its own data and specifies what parts are to be shared and with whom they can be shared.

In general, the CHARM effort covers both operational and software issues across a broad range of projects. Some of these projects involve minor enhancements to existing software systems, while others are developing new systems. However, they all center on a secure distributed middleware solution, called the *CHARM Integration Infrastructure* (CHARM-II). The purpose of this document is to describe the architecture of this infrastructure. We start by giving a brief overview of CHARM-II, describe how it connects existing and future health-care systems, and outline the goals that we established before developing its architecture. We then describe the various components of the CHARM-II architecture.

2. CHARM-II Overview

2.1. System Context

CHARM-II is a middleware system that integrates autonomous and heterogeneous health-care programs so they can share data with minimal impact on existing software. The CHARM-II software, represented by the center box in Figure 1, does not store program-specific data. Rather, it retrieves the requested data from the participating health-care programs in near real-time as it is requested. The outside boxes in Figure 1 represent the information system for some of these participating programs (PP's). Those with solid borders are existing systems; while, those with dashed borders are either currently in development or planned for the near future.

The long-term plan for CHARM includes building Web and PDA interfaces into the system. These program-independent interfaces, represented by the green boxes at the top of Figure 1, will allow authorized users access to shared data without having to be logged into one of participating health systems. As with all use of CHARM-II, access via these additional interfaces will be secured and audited.

The long-term plan also includes the integration of CHARM with systems outside of UDOH. For example, it could integrate the information system for the Division of Child and Family Services (DCFS).

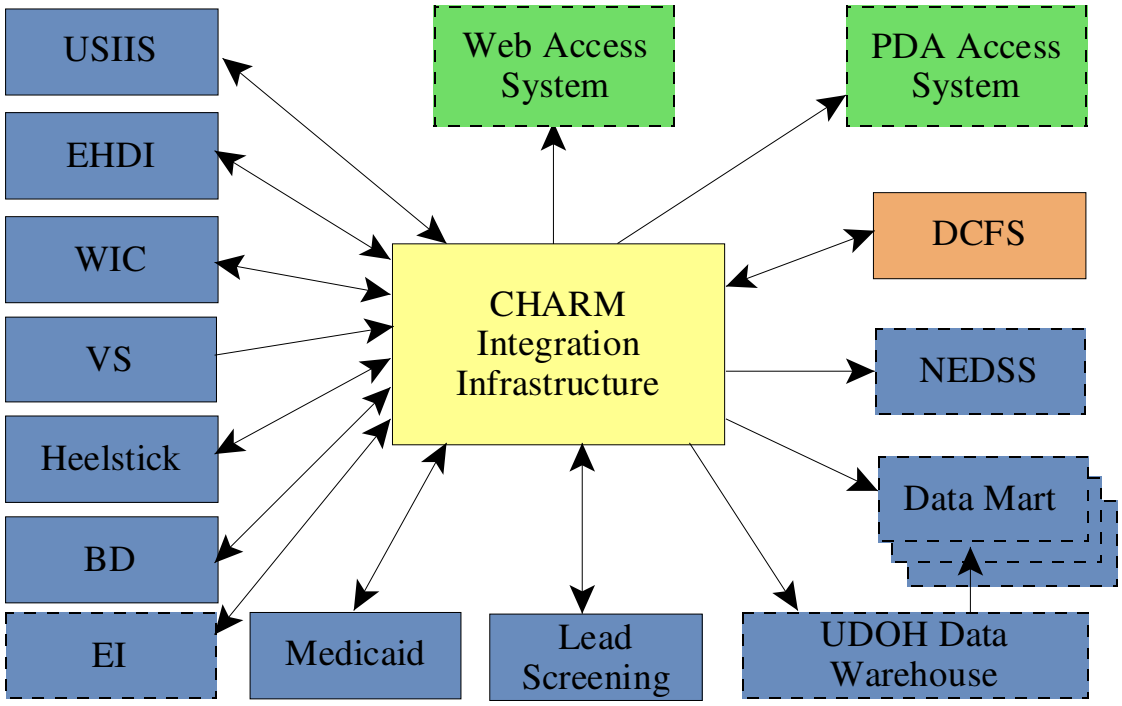


Figure 1 - CHARM Overview

2.2. Goals for CHARM-II

A number of specific goals guided definition of requirements for CHARM-II[1] and eventually its architecture and implementation. These goals fall into five main areas: data access; integration with existing information systems; data ownership and control; privacy, confidentiality and security; and flexibility and extensibility.

Data-Access Goals

- Provide *consistent*, *current*, and *authoritative* information about any child born in Utah or any child who is receiving services from one of the participating health-care programs.
- Establish a catalog of approved program-independent data attributes. This catalog defines which data are shared between programs. We call an instance of these data for a given child a *Child Health Profile* (CHP).
- Provide access to CHP's in real time or near real time, as client-oriented activities are being performed, i.e., at the time of encounter.
- Allow users to match children as they are entered in a PP's information system with persons already known to CHARM.
- Alert users of exceptional conditions for a child as they access that child's CHP. These conditions can be set or removed by any PP and should be documented in a manner that does not require users to interpret another program's data.
- Provide efficient transfer of large sets of data between programs. For example, a data warehouse could retrieve all recently submitted immunization data from USIIS.

Integration with Existing Information Systems

- Allow any program that wants to participate to do so with minimal impact upon existing information systems.
- Minimize the impact of real-time data sharing to the performance of a PP's information system.
- Reduce or eliminate the re-entry of common demographic information that can be obtained from the CHP for children already known to CHARM.
- Allow PP's to maintain and enhance their own information systems independent of CHARM or any other PP.

Data Stewardship

- Ensure that PP's retain stewardship of their own data.
- Allow a PP to define which of its own data it is willing to share with others and describe the data's intended use and meaning.
- Allow a PP to define security policies that govern who has access to its data.

Privacy, Confidentiality and Security

- Ensure that clients (parents or guardians of the children known to CHARM) can choose who can and cannot view the data in their CHP's.
- Ensure that all program and department security policies can be properly enforced.

[1] The functional requirements for CHARM-II are given elsewhere, namely in the "*Requirements Definition for the Child-Health Advanced Record Management Integration Infrastructure"*

- Keep an audit of who accessed what data, at a CHP, user, and/or data-attribute level.

Flexibility and Extensibility

- Allow programs wanting to participate to "plug-in" with relative ease.
- Allow CHARM to grow in scope to include children up through age eighteen, and maybe even beyond.
- Allow CHARM to grow in size (i.e. data and transaction volume) to serve a large percentage, and ideally all, of the health professionals in the state.
- Eventually, provide clients with secure access to their own information.

2.3. Architectural Overview

CHARM-II is not a single piece of software. Rather, it is itself a collection of distributed components, each serving a purpose and fulfilling a specific set of requirements. The major components are as follows:

- a central server, called the *CHARM Server*
- an agent for each PP, called a *CHARM Agent*
- an alert-generation engine for each PP

The CHARM Server is responsible for optimizing and executing all queries for the child-health care data. To support this primary activity, it also uses and manages a catalog of data definitions and other configuration parameters. It enforces security policies established by the individual PP's, keeps an audit trail of all queries, and monitors overall performance. Section 3 describes the CHARM Server in more detail.

A CHARM Agent for a PP acts as both a client and a server. As a client, it forms queries on behalf of its PP and submits those queries to the CHARM Server. When it receives the results of a query, it translates them into a form that is most suitable for the existing software of the PP. As a server, a CHARM Agent provides services to the central CHARM Server for extracting information that others might request. Each program is responsible for deciding which of its own data will be shared and what other programs will be allowed to access that data. Section 4 provides additional details on CHARM Agents.

Every PP can have its own alert engine that generates alert messages for children that need attention. A program can define its own rules or conditions that cause the generation of alerts. The alert engine checks those conditions for children as their records change or on a periodic basis. See Section 5 for more information.

If we look at all the services provided by all the PPs, the collective information available through these services represents a virtual health record for any given child known to the CHARM system. We refer to this virtual record as a *Child Health Profile* (CHP) and the database that defined its structure as the *Catalog*. The formation and processing of queries will be based on this catalog. The Catalog is stored in a database, because we anticipate that it will change over time. In fact, there may be several versions of the Catalog in use at any given time since the PP's may update their information systems as they see fit and not on a synchronized schedule. A

conceptual model, called the *CHARM Meta-Model*, defines the structure of the Catalog database. Both the CHARM Meta-Model and the Catalog are fully described with UML models.

To coordinate information among the PP's, the CHARM Server will manage a small amount of demographic data for each person known to the system. The part of the server that does this can be thought of as a special kind of PP and, therefore, will have its own agent. We will refer to this special agent as the *Core Agent* and we will call the data that it manages the *Core Data*. The content and structure of the Core Data are defined by a UML model called the *CHARM Core Data Model*.

3. CHARM Server

3.1. Use Scenarios

Before describing the details of the server, walking through a few of complete use scenarios will help set the stage. Figure 2 illustrates the most common type of use scenario – the execution of a query. In this particular scenario, Early Intervention (EI) sends a query to its own agent that requests the immunization history and the primary-care physician for a child. The agent for EI translates the raw query and maps the EI's ID for the child subject to a program-independent CHARM ID. It then sends the query to the CHARM Server, and more specifically, the *Query Manager* component of the CHARM server. The Query Manager executes the query by determining where to find the requested information, retrieving it from other PPs (in this case *USIIS* and *Early Hearing Detection and Intervention*), and then assembling a final result from what those programs send back. The Query Manager then sends this final result back to the EI Agent which in turn makes it available to the EI program.

Figure 3 shows three other common use scenarios for merging, adding, and delete CHP's. The first one shows an EI user's find request, which returns a list of possible matches. After reviewing the possible matches, the user decides to merge two of them, so a merge request is

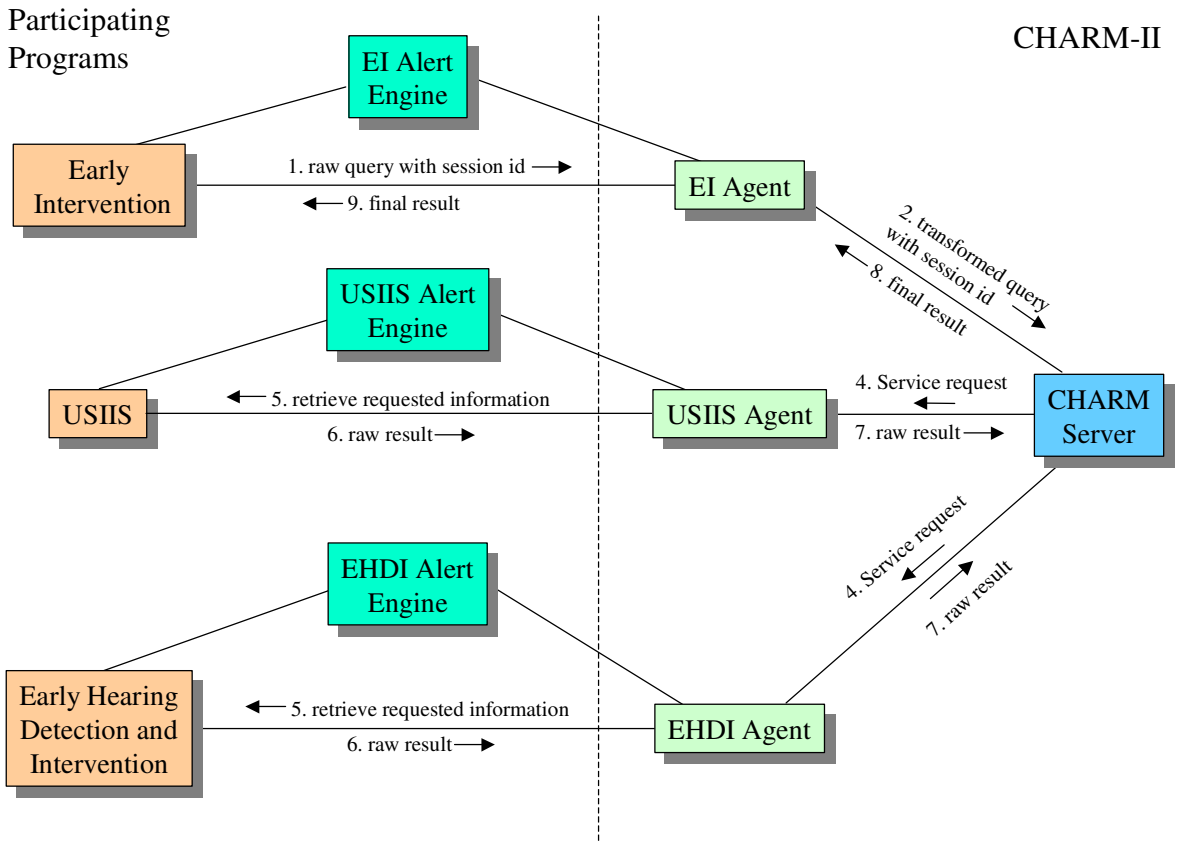


Figure 2 - Overview of a query processing scenario

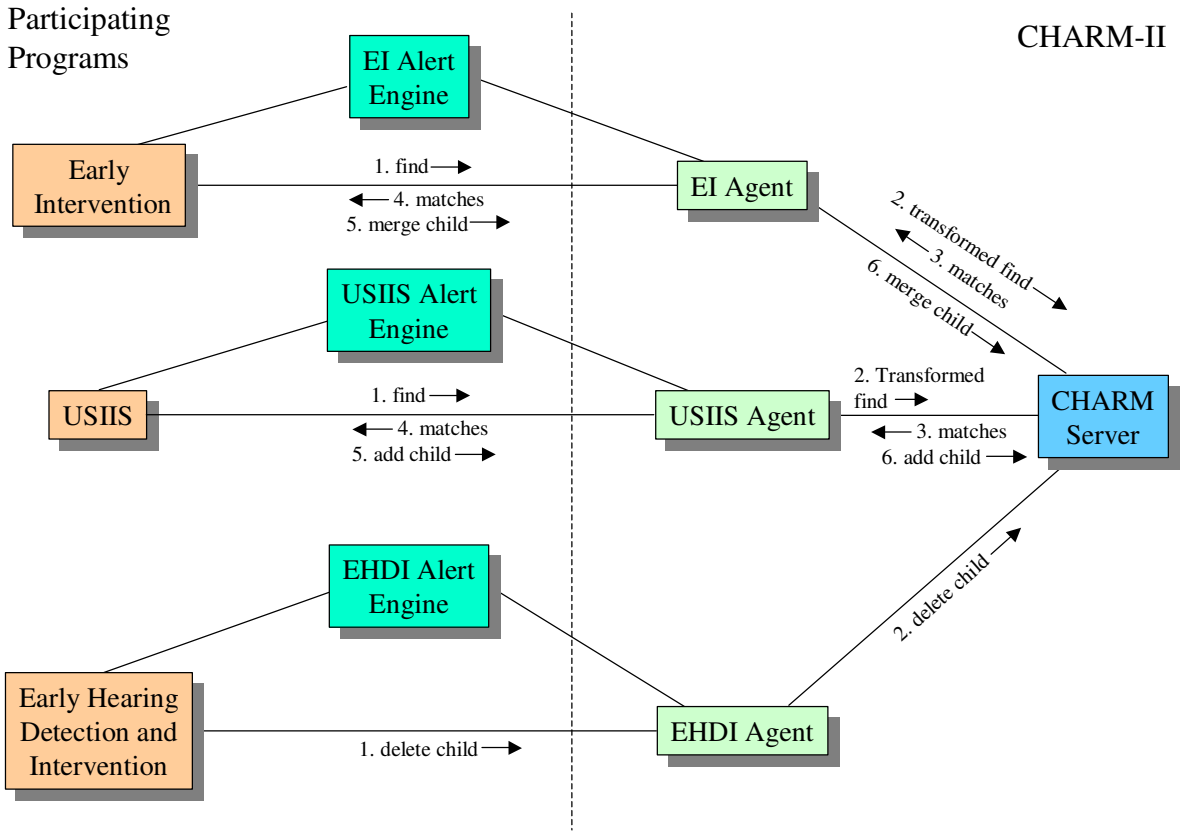


Figure 3 - Finding, merging, adding, and deleting Child Health Profiles

sent to the EI Agent, which translates and forwards it to the CHARM server. The second scenario (shown in the middle of the figure with the USIIS program) is very similar, except the user decides to add a new CHP. The third scenario shows the flow of messages for a delete request.

All these types of requests, as with a query, involve the PP sending messages to its own agent. The agent translates that message into a standard format and maps program-specific ID's to CHARM ID's and then forwards the message to the CHARM server. The server completes the request and returns the results to the agents, which makes them available to the PP.

Figure 4 shows a different kind of use scenarios involving the generation of alerts. In this case, the PP notifies its alert engine that the data for a child has changed. The engine may then request additional information about that child from the PP. Next, it checks to see if that child meets the condition of any alert-generation rule identified for that PP. If a child satisfies the condition, then the alert engine generates an alert as specified by the rest of the rule and sends it to the PP's agent. The new alert has to go through the agent, so program-specific data can be translated to program-independent data. It also helps keep the interface to CHARM-II simple and secure because the agents are the only components allowed to talk with the server.

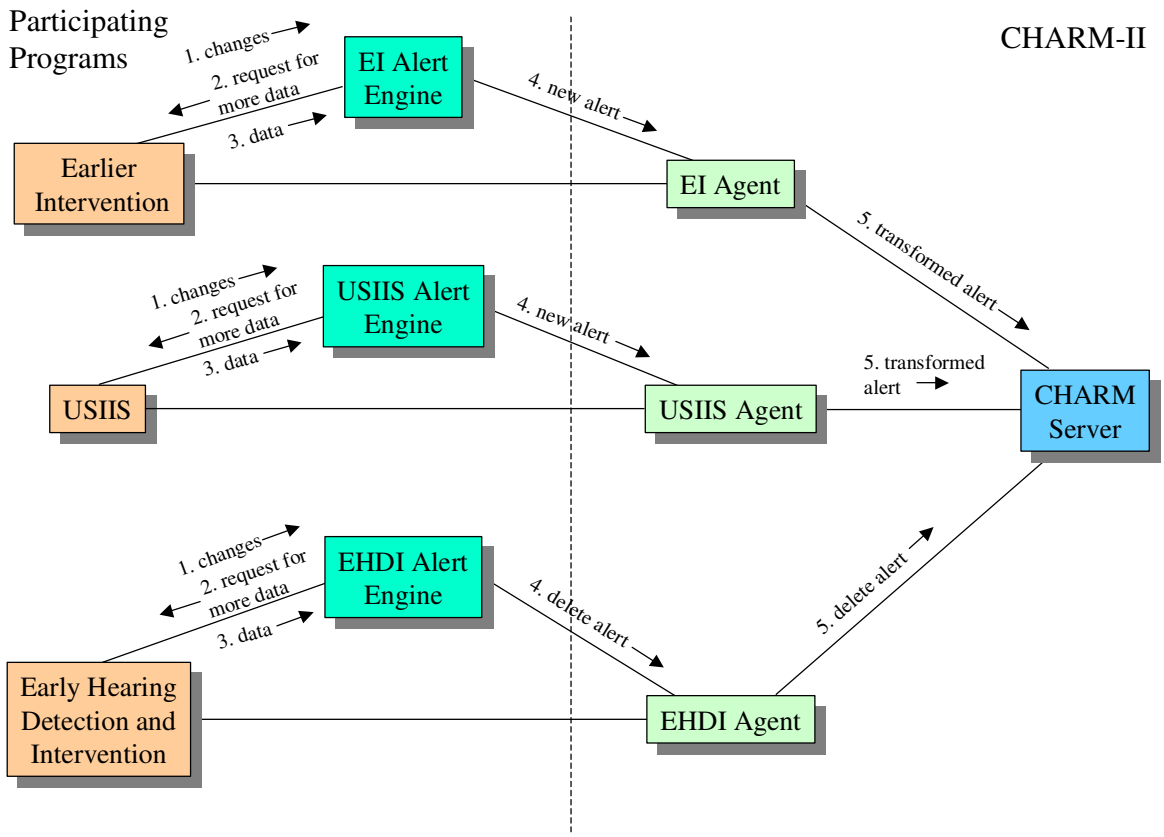


Figure 4 - Overview of several alert processing scenarios

3.2. CHARM Server Components and Their Interactions

The CHARM Server consists of a number of independent components that can be distributed or replicated as the need arises. Figure 5 provides a high-level view of these components and Figures 6 - 10 show how they interact for five use scenarios.

Query Processing

As Figure 6 illustrates, when a query is sent to the server, it first goes to the Query Manager (*message 1*). The Query Manager checks authorization for the user by passing the session id sent with the message to the Security Manager, which in turn retrieves user profile information from Siteminder. (See *messages 2* and *2.1*.) Note that Siteminder is not part of CHARM-II. It is the software that the state ITS department has chosen for managing users and access rights. The Security Manager returns the user profile to the Query Manager. If the session is valid, then Query Manager retrieves the query definition from the Catalog (*message 3*) and compares the user's profile against the required access rights for the request query. The Query Manager will remove anything from the query definition that the user is not allowed to view.

The query definition consists of a query data model, execution strategy, and result-building strategy. The data model describes the structure of the data that the query will return. The

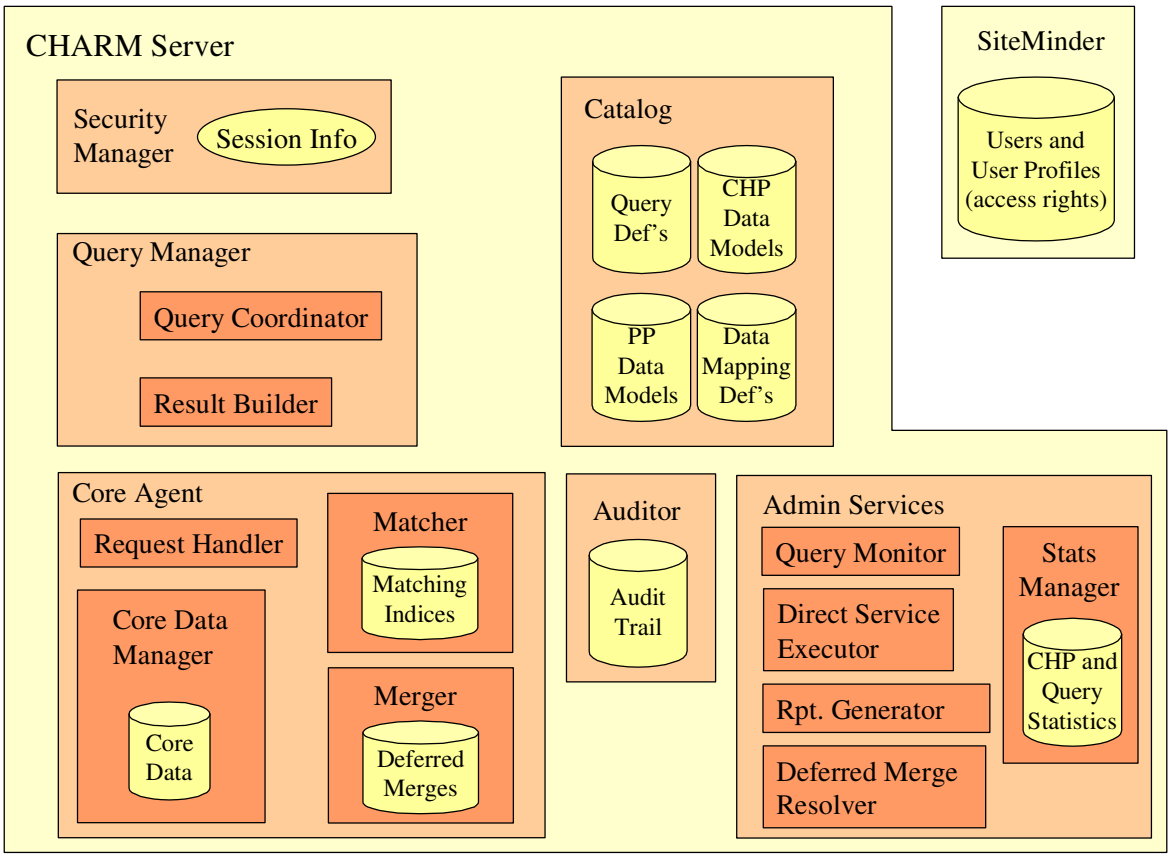
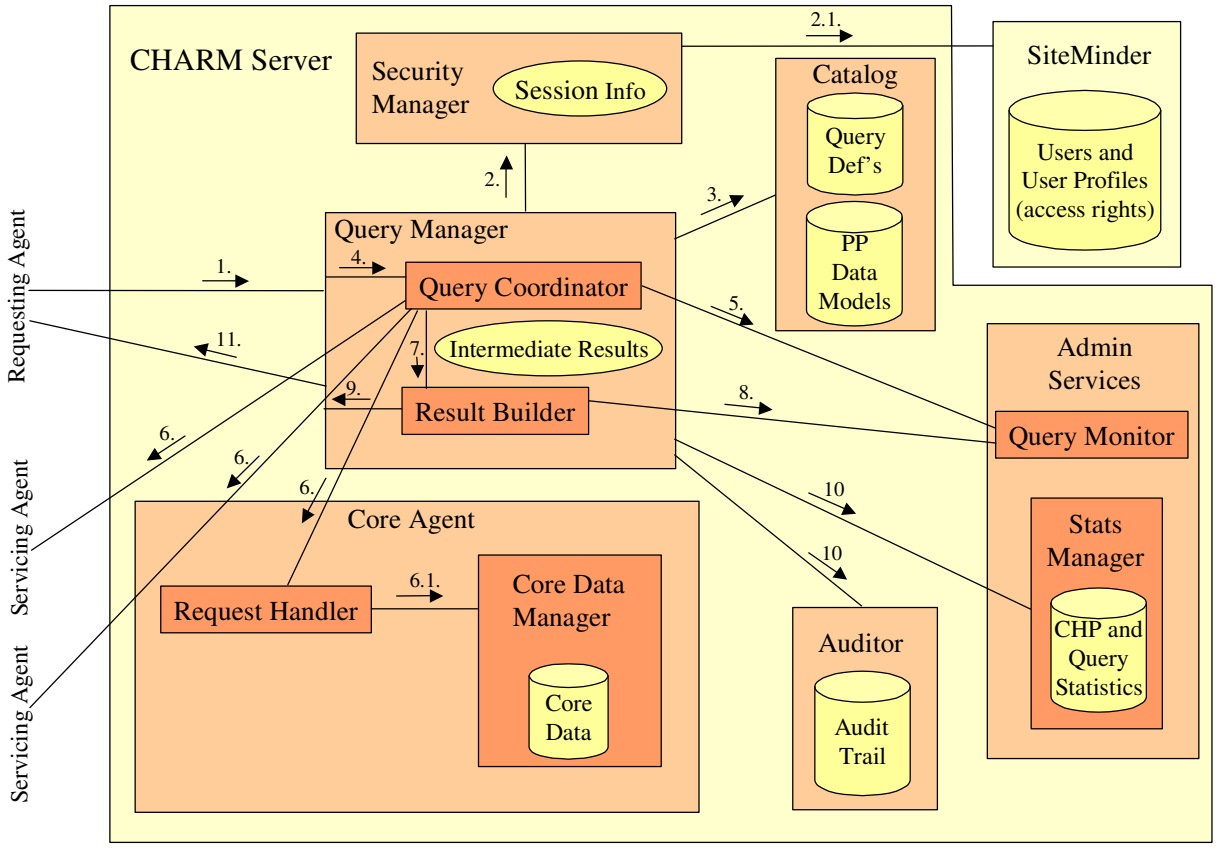


Figure 5 – CHARM-II Version 1 Server Components

execution strategy describes what services are needed to retrieve the information and the order in which they should be called. In general, an execution strategy is a tree of actions where each action is either a service call or a list of other actions. Lists can differ in how their actions are executed. For some lists, the actions are all executed concurrently; for others, the actions are executed sequentially. Also, some lists require all their actions to be completed; whereas, others only require one action to be completed. The result-building strategy of a query definition describes how the immediate results returned by the services are combined into a final result. Like a query strategy, a result-building strategy is a tree of actions. However, the actions for a result-building strategy specify data transformations instead of services.

In a future version of CHARM-II, the Query Manager will include another component, namely, an optimizer for constructing the execution and result-building strategies on the fly for ad-hoc queries.

After retrieving a query definition from the catalog and pruning it according to the user's access rights, the Query Manager sends it to the Query Coordinator (*message 4*). The Query Coordinator sends a status message to the Query Monitor (if it is running) and then processes the execution strategy (*messages 5* and *6*). As intermediate results come back from the services, it saves those results with the query. Note that the Core Agent in the server can provide certain kinds of information (mostly demographic information) just like any other agent.



Messages

1. query(Query Name, Translated Parameters Session Id)
2. validate(Session Id) return success/failure and User Profile
2.1 lookup(User Name) returns User Profile
3. lookup(Query Name) returns Query Definition
4. process(Query)
   A Query object is created by Query Manger after its looks the query definition in the Catalog, but before sending this message. The Query object represents a query in progress and includes a definition, as well as state information.
5. record(Query)
   Note that the status at this point is "In progress"
6. serviceInvocation(service name, parameters) returns service results
7. buildResult(Query)
8. record(Query)
   Note that the status at this point is "Complete", "Partial", or "Failure"
9. returns Query with CHARM Result
10. record(Query)
11. returns CHARM result

Figure 6 – Query Process from a Server Perspective

Once the Query Coordinator has collected the necessary information or the allotted time elapses, the Query Coordinator passes the query (and its definition) on to the Result Builder (*message 7*). The Result Builder processes the result-building strategy to combine the intermediate results into a final result. When it is finished, it sends a status message to the Query Monitor (if it is running) and sends the final result back (see *messages 8-11*). Note the Query Manager may send statistics and audit trail information to the Stats Manager and Auditor as it returns the final result, if those features are enabled.

Finding and Merging Health Care Profiles

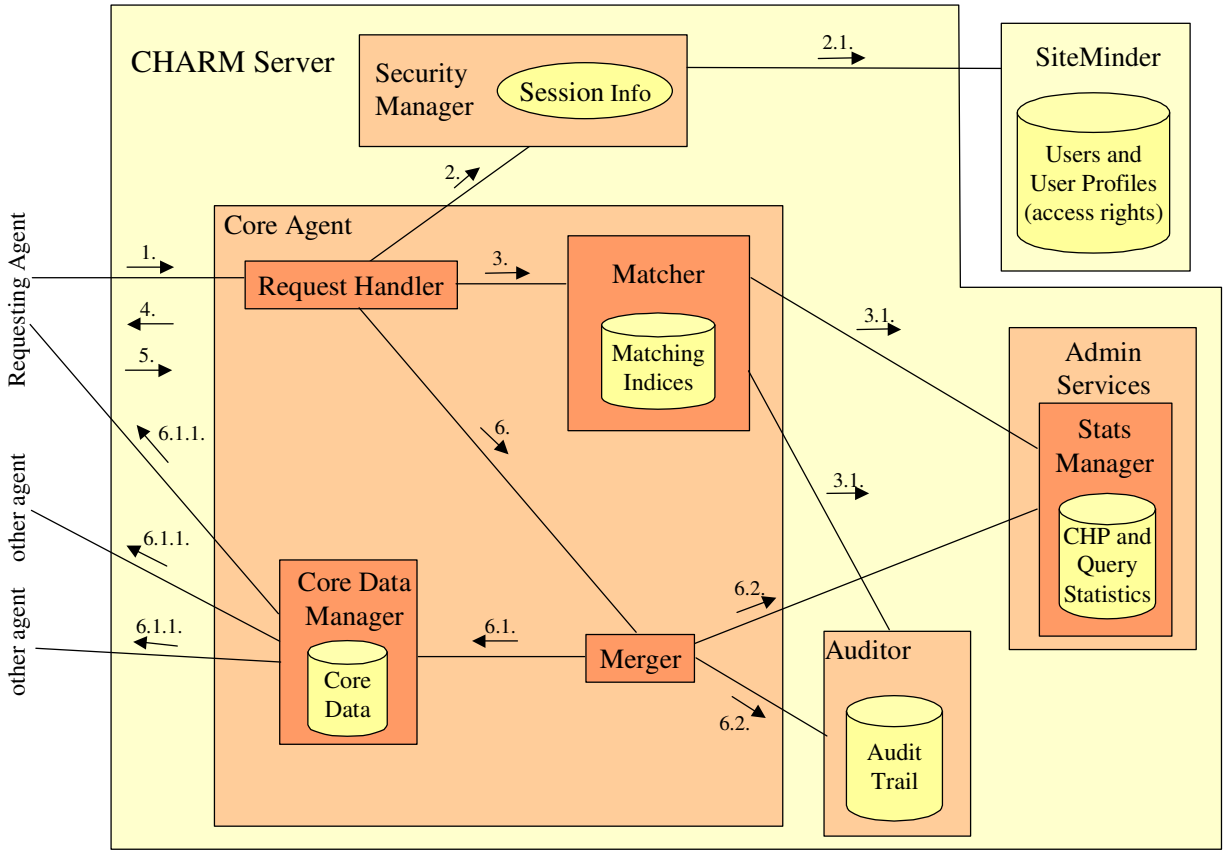
From time to time, a PP needs to add new children to its own system and therefore to CHARM. As part of this process, a user can search CHARM to see if there is already a CHP for the child. If there is one, the user can choose to simply use that CHP. If for some reason there is more than one CHP for the same child, the user can merge them into one.

Figure 7 shows the flow of messages involved in finding and merging health CHP's. First, the PP's agent sends a "find" message to the Core Agent in the server (*message 1*). Next, the Core Agent checks the validity of the session with the Security Manager and passes requests onto the Matcher (*messages 2*, *2.1*, and *3*). The Matcher searches for similar CHP's based on the parameters contained in the find message and creates a set of possibilities ordered "most-likely first." It also records statistics and audit trail data if those features are turned on. Note that these messages are both labeled 3.1 in Figure 6 because they can happen concurrently. The Matcher returns the set of possibilities to the requesting agent (*message* 4), which in turn returns it to the PP.

At this point, this point the PP does whatever it needs to do with the set of possibilities. In most cases, it would display them to the user and let the user decide whether to use an existing CHP, merge two or more existing CHP's, or add a new one.

If the user decides to merge two or more existing CHP's, the PP sends a merge message back through its agent (*message 5*). This merge request will contain information about which fields to keep from each of the original CHP's. The Core Agent will receive this message, validate the session, and pass the message on the Merger (*messages 2* and *6*). The Core Agent will complete the merge by changing the necessary information in the core database (*message 6.1*) and retiring one of the CHARM ID's. All future requests to the old ID will be forwarded to the other ID. To do this, the Core Agents notifies all the agents of a change in the ID mappings (see the *6.1.1 messages*). Statistics and audit trail information are saved if these features are turned on (see *6.2 messages*).

The addition of a new child to the system is very similar to the above scenarios, except that an "add" message is sent to the Core Agent instead of a "merge" message.



Messages

1. find(FindObject, Session ID)
   Note that the FindObject includes child search criteria
2. validate(Session ID)
2.1 lookup(User Name)
3. lookup(FindObject) returns FindObject
   Note that the possible matches are added to the FindObject
3.1 record(FindObject)
4. return FindObject
5. merge(MergeObject, Session ID)
   Note that the MergeObject includes all the necessary merge specifications
2. validate(Session ID)
6. merge(MergeObject)
6.1. merge(MergeObject)
6.1.1. merge(MergeObject)
6.2. record(MergeObject)

Figure 7 – Finding matches and Merging Child-Health Profiles

Deferring Merges

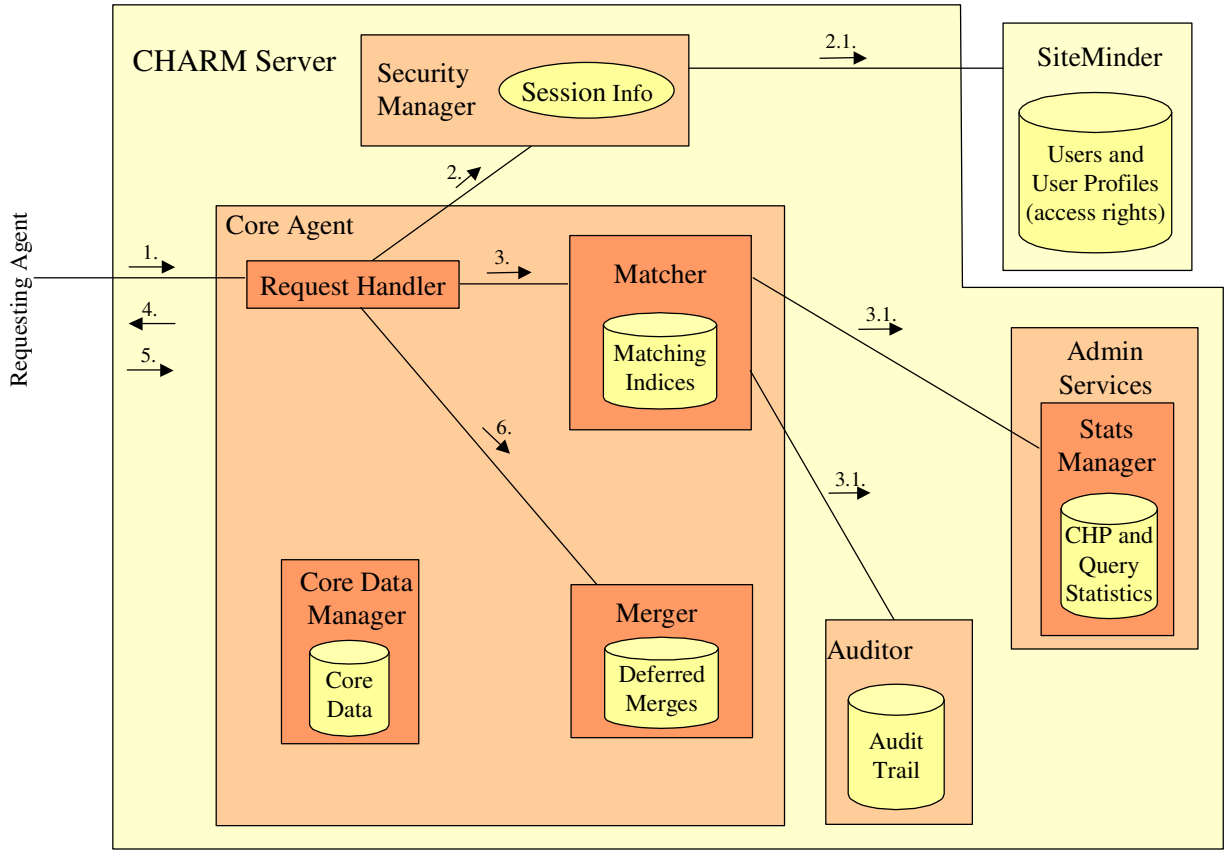
In some cases, a user may suspect that two existing CHP's are the same, but isn't quite sure. In these cases, the user can defer the merging of CHP's until someone else can do some additional research and resolve the merge request. The same process can be used when a new child needs be added to the system and the user suspects that an existing CHP is for the same child, but is not sure. However, in this case, a new CHP is added for the child and the deferred merge is submitted for that CHP and the suspected duplicate CHP. Figure 8 illustrates the flow of messages for finding matches and them deferring a merge. Figure 9 shows the resolution of the deferred merges.

Automatic Duplicate Detection

When the Matcher is not processing requests from agents, it automatically scans the core data for duplicate CHP's. In a nutshell, it selects a subject CHP and tries to find matches for that CHP. If some possibilities are found, it records a deferred merge with the Merger. Later, someone will make final decisions about the possible matches and resolve those deferred merges.

Adding or Deleting Alerts

The processing of "add" or "delete" alert requests is relatively straightforward. As shown in Figure 10, a requesting alert engine sends an "add" or "delete" message to the Core Agent. The Core Agent validates the session (*messages 2* and *2.1*) and sends the request to the Core Data Manager (*message 3*), which makes the change to the core data. The Core Data Manager also records some statistics via the Stats Manager (*message 4*), if necessary.



Messages

1. Find(FindObject, Session ID)
   Note that the FindObject includes child search criteria
2. validate(Session ID)

2.1 lookup(User Name)

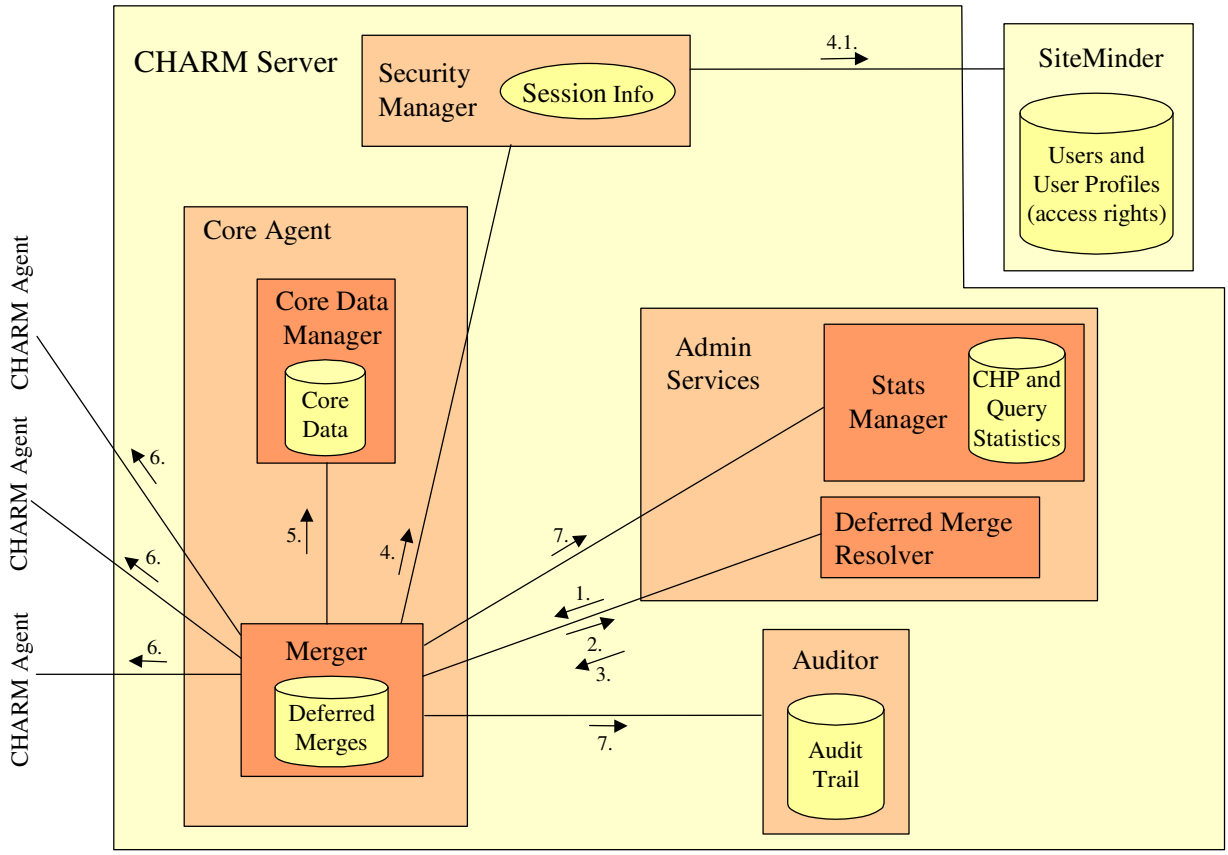
3. lookup(FindObject) returns FindObject
   Note that the possible matches are added to the FindObject

3.1 record(FindObject)

4. return FindObject
5. deferMerge(MergeObject, Session ID)
   Note that the MergeObject includes all the necessary merge specifications

2. validate(Session ID)
6. deferMerge(MergeObject)

Figure 8 – Finding matches and deferring merges

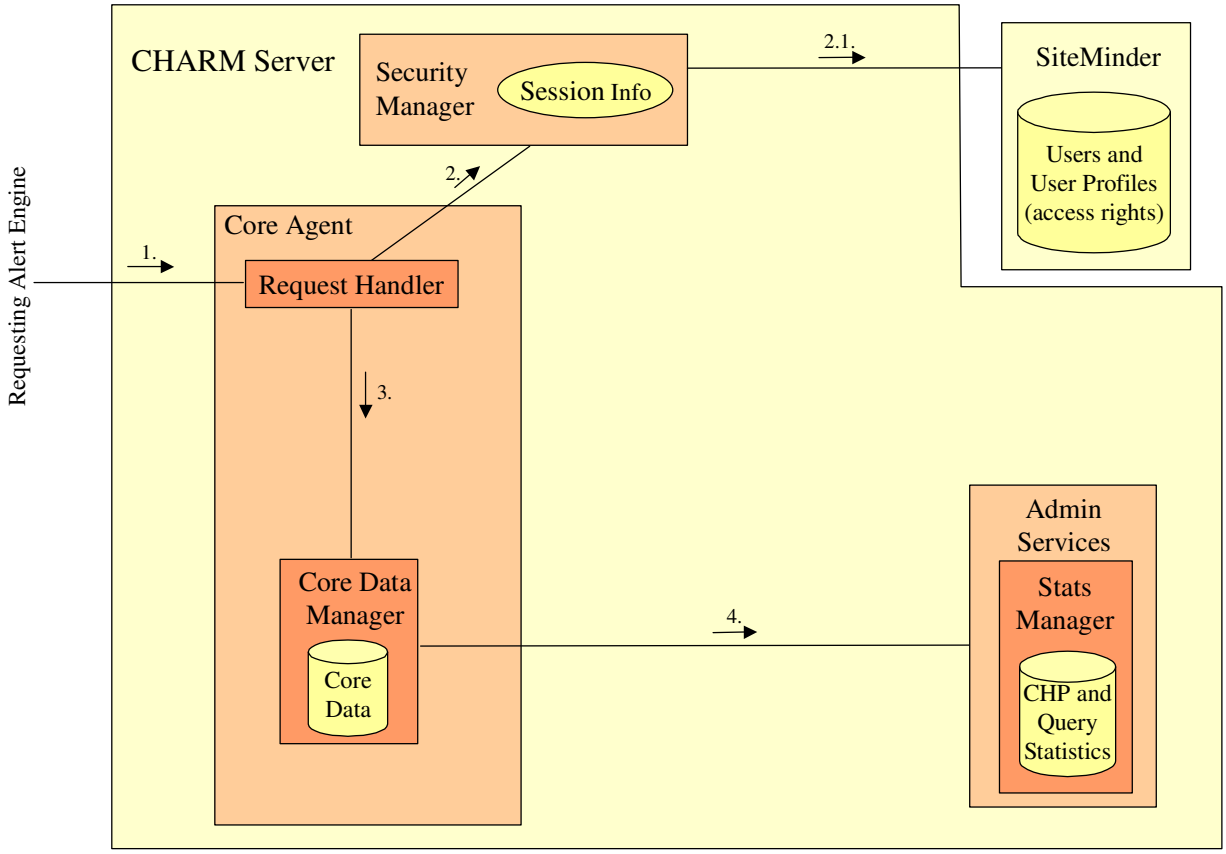


Messages

1. getNextDeferredMatch ()
2. return MergeObject

   Note that the Deferred Merge Resolver will need to perform some queries to retrieve enough data for the two candidate CHP so the user can compare them and decide what (if anything) to merge.
3. merge(MergeObject, Session ID)
4. validate((Session ID)
5. merge(MergeObject)
6. merge(MergeObject)
7. record(MergeObject)

Figure 9 – Resolving deferred merges



Messages

1. addAlert(Alert, Session ID) or deleteAlert(Alert, Session ID)
   Notes that the Alert object include a CHARM ID, alert type, and alert message.
2. validate(Session ID)
3. addAlert(Alert) or deleteAlert(Alert)
4. record(Alert)

Figure 10 – Adding or Deleting Alerts

4. CHARM Agents

4.1. Agent Components

Like the CHARM server, an agent consists of several independent components. However, unlike the server, an agent cannot be distributed or replicated. All the components for a given agent will run on a single system. Some agents will run on the same system as some components of the CHARM server. Other agents will run on a PP's system; and still others might run on an independent machine.

Figure 11 shows the six primary components of the CHARM agent: *PP Interface*, *Request Processor*, *ID Mapper*, *Catalog Cache*, *Service Processor*, and *Admin Interface*. The *PP Interface* is actually a collection of technology-specific program interfaces for communicating with PP information systems. For example, the *RMI PP Interface* provides remote methods for submitting queries and the retrieving the results of those queries. The *Request Processor* is responsible for translating raw requests into program-independent requests, sending that them to CHARM server, and tracking they completion. It actually consists of two subcomponents: one for handling queries and for handling all other types of requests (finding, merging, adding, and deleting CHP; adding and deleting alerts, etc.) The process of translating a raw request into a program-independent request involves mapping PP ID's to CHARM ID's. The Request Process

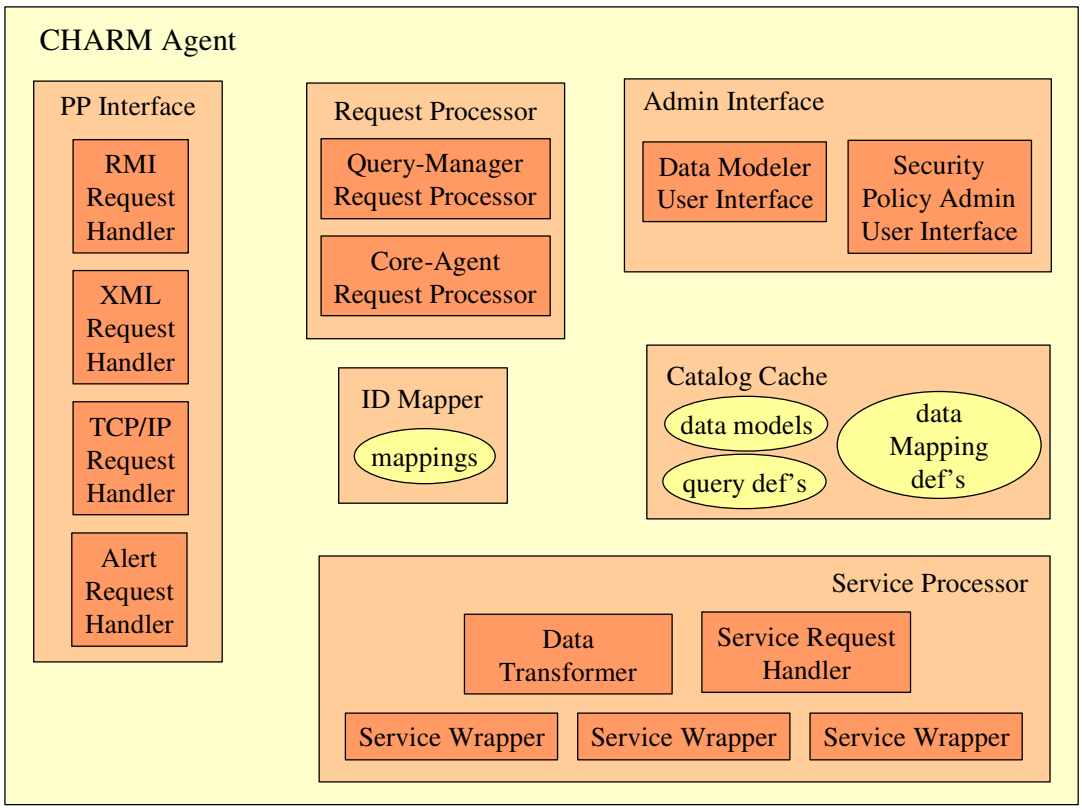


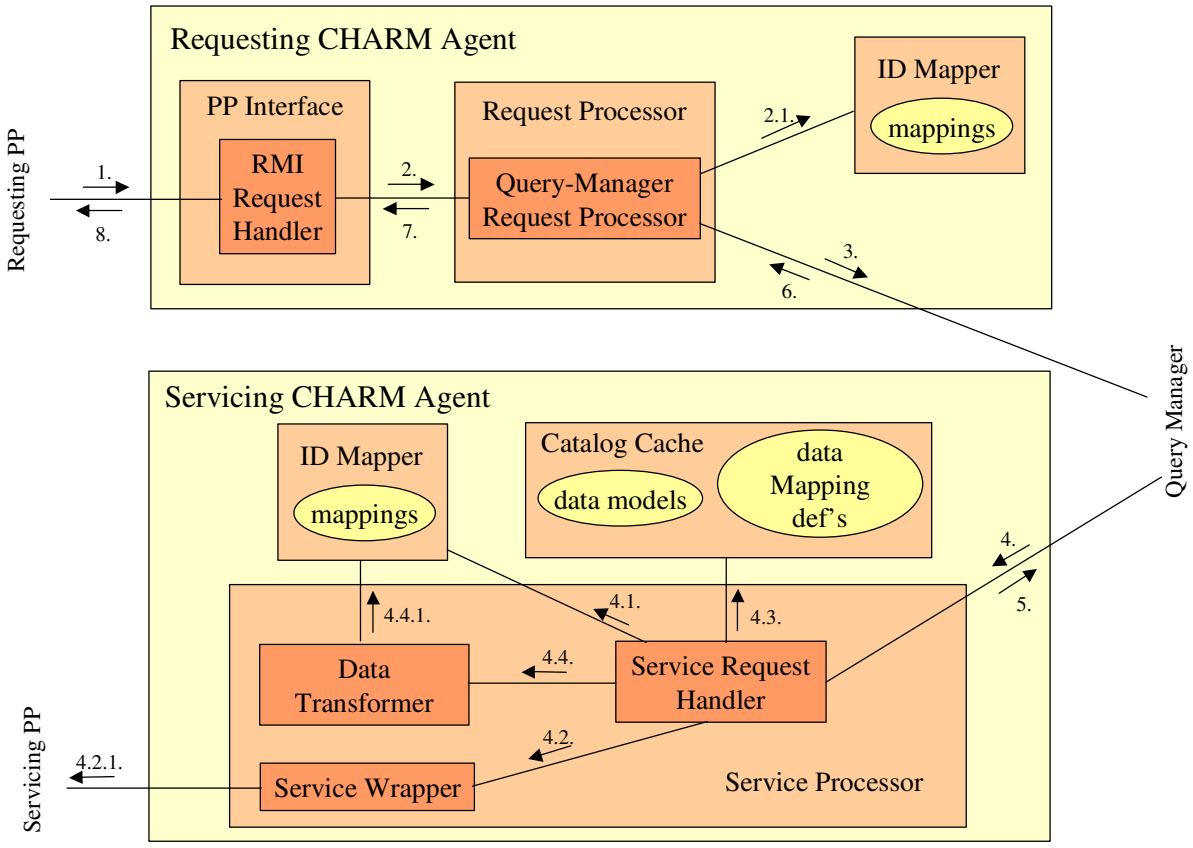
Figure 11 – Overview of Components the Comprise a CHARM Agent

uses the *ID Mapper* to do this. It may also need information about the either the structure of PP-specific data or program-independent data to do the translation. The Request Processor can access a local *Catalog Cache* to get the necessary meta-data. The *Service Processor* is responsible for handling all service invocations coming from the CHARM Server. Like the Request Processor, it can access the Catalog Cache to get any meta-data that it needs to invoke specific PP services or perform data translations.

4.2. Agent Interactions

Figure 12 shows the processing of a typical query from an agent's perspective. The requesting program sends a raw query to its agent via one of the PP Interface modules (*message 1*). In this case, the requesting process uses the RMI PP Interface. The RMI PP Interface forwards the request onto the *Query Processor* of the Request Processor (*message 2*), which translates the request and maps the PP ID to a CHARM ID (*message 2.1*) and sends the translated query to Query Manager in the CHARM server. As explained in the previous section, the Query Manager decides where to get the requested data and invokes the necessary services by sending messages to *Service Request Handlers* of one or more other PP Agents (*message 4*). When a Service Request Handler in a PP Agent receives service invocation message, it maps CHARM ID's to PP ID's (*message 4.1*), calls the request PP service via a *Service Wrapper* (*message 4.2*), looks up data mapping definitions in the local Cache Catalog (*message 4.3*), and then translates the resulting data from the service call using the mapping definitions (*message 4.4*). Finally, the Service Request Handler sends its results back to the Query Coordinator (*message 5*) which then combines it with other intermediate results and send the final CHARM result back to the requesting agent (*message 6*) and requesting PP (*messages 7 and 8*).

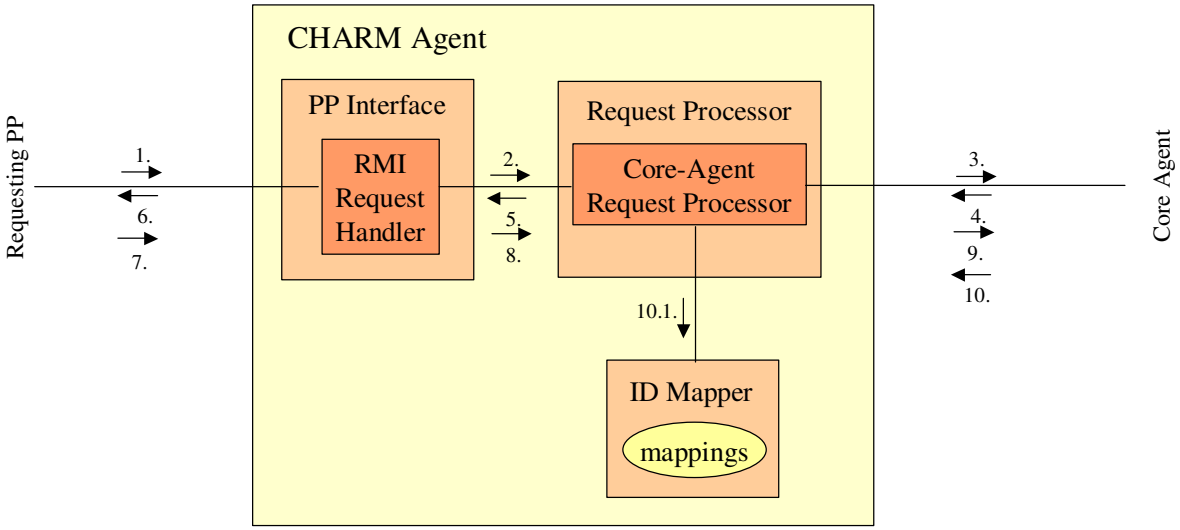
Figures 13-15 illustrate three other use scenarios from an agent perspective, namely the finding and merging of child-health profiles, deferring merges, and adding alerts.



Messages

1. query(Query Name, Parameters, Session Id)
2. query(Query Name, Parameters, Session Id)
2.1 translate(Parameters)
3. query(Query Name, Translated Parameters, Session Id)
4. serviceInvocation(service name, parameters)
4.1 mapToPPID(CHARM ID) return PP ID
4.2 call<<service name>>(Translated Parameters) returns result
4.3 lookupDataMappings(service name) returns data mappings
4.4. translate(result, data mappings) returns service result
5. returns service result

Figure 12 – Query processing from a CHARM Agent perspective

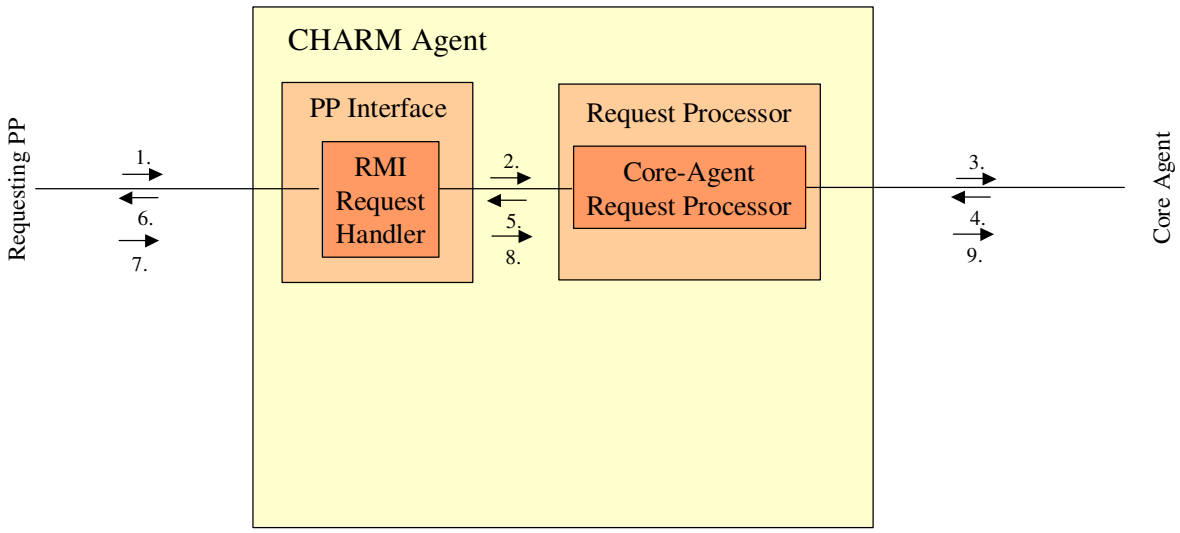


Messages

1. find(Child data, Session Id)
2. find(Child data, Session Id)
3. find(FindObject, Session Id)
   Note that the FindObject includes child search criteria
4. returns FindObject
   The FindObject coming back from the Server includes Possible Matches
5. returns FindObject
6. returns FindObject
7. merge(Merge specification, Session Id)
8. merge(Merge specification, Session Id)
9. merge(MergeObject, Session Id)
10. merge(MergeObject)

10.1. changeIdMapping(CHARM ID 1, CHARM ID 2)

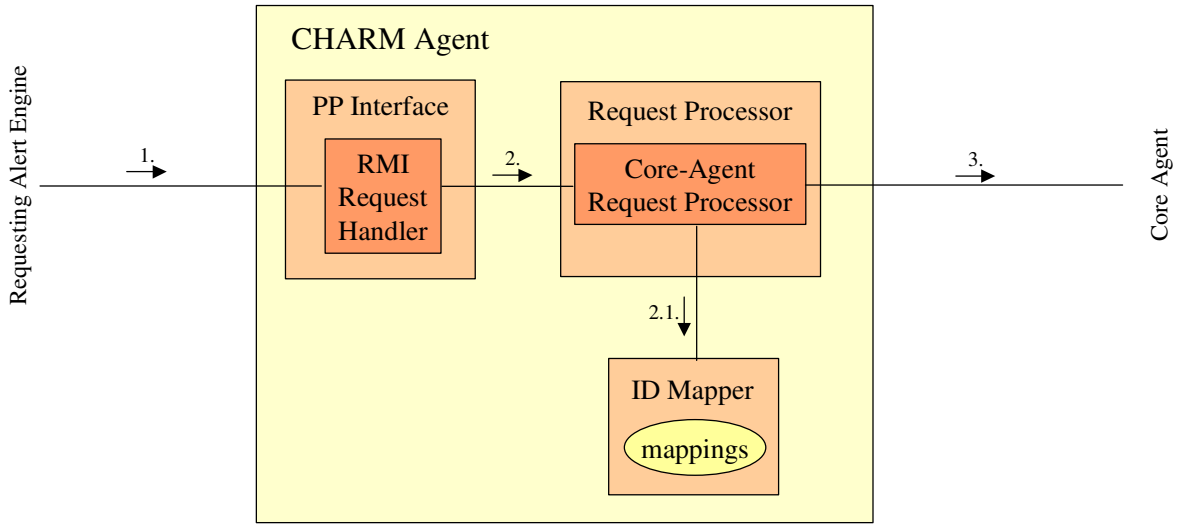
Figure 13 – Finding matches and merging from a CHARM Agent perspective



Messages

1. find(Child data, Session Id)
2. find(Child data, Session Id)
3. find(FindObject, Session Id)
   Note that the FindObject includes child search criteria
4. returns FindObject
   The FindObject coming back from the Server includes Possible Matches
5. returns FindObject
6. returns FindObject
7. defer_merge(Merge specification, Session Id)
8. defer_merge(Merge specification, Session Id)
9. defer_merge(MergeObject, Session Id)

Figure 14 – Finding matches and deferring merges



Messages

1. add(Alert, Session ID)
   Note that at this point, the Alert object includes a PP ID
2. add(Alert, Session ID)

2.1 mapToCHARMID(PP ID) returns a CHARM ID
   The request process will replace the PP ID in the Alert object with the CHARM ID

3. add(Alert, Session ID)

Figure 15 – Adding alerts from a CHARM Agent perspective

5. Alert Engine

Each PP has its own version of the Alert Engine with it own set of custom alert-generation rules and configuration parameters. However, all the Alert Engines have the same basic components shown in Figure 16. These include a *PP Interface, Alert Generator, Rule Manger, Service* Processor, and Admin *Interface*. The *PP Interface* is a collection of program interfaces for communicating with the PP. For example, when a child's information changes in a PP's information system, it can sent a change notification to its Alert Engine via one of these interfaces. Depending on the Alert Engine mode of operation, this notification may trigger the generation of new alerts or the removal of old alerts. The *Alert Generator* is responsible for analyzing the status of a child with respect to the conditions stated in the *Alert-Generation Rules* and then creating or deleting alerts as needed. To access the rules, it communicates with the *Rule Manager* and to access additional information about a child it communicates with its own *Service Request Processor*. Once it decides to create or delete an alert, it does so by sending a request to the PP's Agent via the *Agent Interface*.

An Alert Engine can be configured to run on several different modes: *immediate*, *deferred*, and *periodic*. In the *immediate mode*, the PP information system notifies the Alert Engine whenever

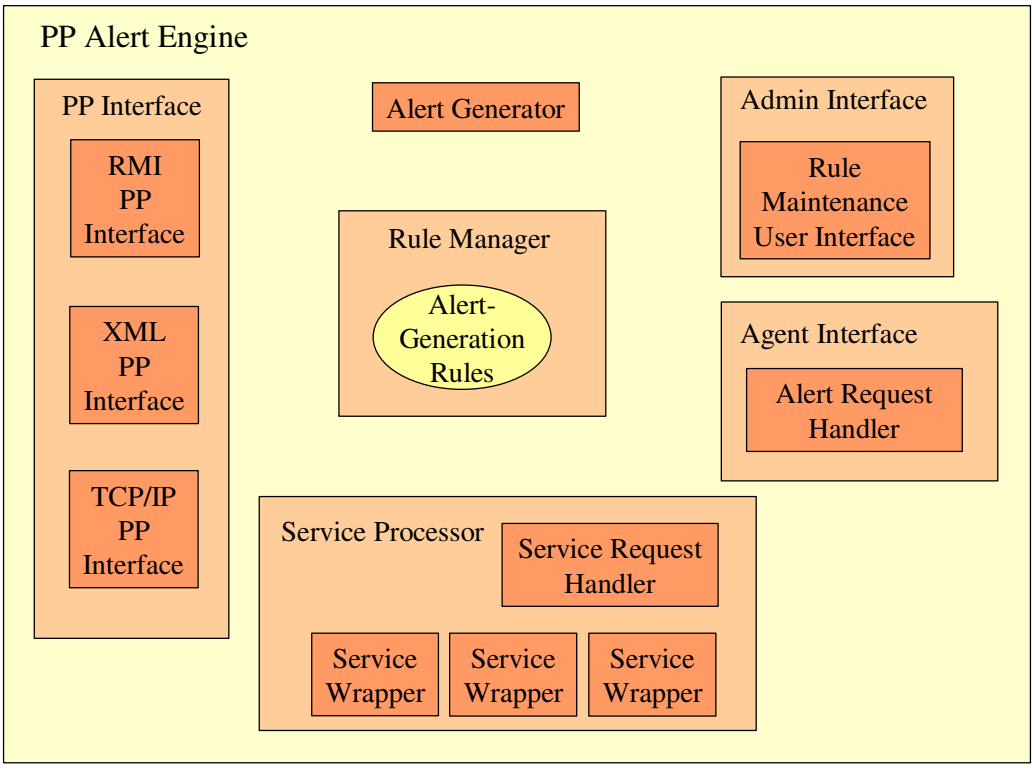


Figure 16 – Overview of Alert Engine components

a child's data changes. The Alert Engine immediately starts a process to check the status of the child with respect to the alert conditions and to create/delete alerts as needed. In the *deferred mode*, the PP information system still notifies the Alert Engine whenever a child's data changed, but the Alert Engine stores that notice in a queue. Then at a later time (like midnight), it processes all of the changes notices together. In the *periodic* mode, the PP information system does not notify the Alert Engine of changes. Rather, the Alert Engine periodical reviews all children known to the PP with respect to the alert conditions and creates/deletes alerts as needed.